1. What are the new features introduced in Java 17?

Answer: Java 17 introduced several new features and enhancements, including:

Pattern Matching for instanceof (JEP 394): Simplifies the common pattern of combining instanceof and type casting.

Sealed Classes (JEP 409): Provides a more declarative way to restrict which classes or interfaces can extend or implement them.

Records: Provides a compact syntax for creating immutable data classes

Text Blocks: Allows multi-line string literals without escape characters - Improved Random Number Generators: Provides a new interface and implementations for random number generation

Switch expressions to handle multiple cases: Switch expressions in Java 17 provide a concise way to handle multiple cases. Unlike traditional switch statements, they can return a value and be used in a more functional style.

Enhanced Pseudo-Random Number Generators (JEP 356): Adds new interfaces and implementations for pseudorandom number generators (PRNGs).

Deprecation of the Applet API (JEP 398): Marks the Applet API for future removal.

Removal of the Experimental AOT and JIT Compiler (JEP 410): Removes the experimental ahead-of-time (AOT) and just-in-time (JIT) compiler.

Strong Encapsulation of JDK Internals (JEP 403): Strongly encapsulates JDK internals, except for critical internal APIs such as sun.misc.Unsafe.

Foreign Function & Memory API (Incubator, JEP 412): Introduces an API for interacting with code and data outside of the Java runtime.

Vector API (Second Incubator, JEP 414): Introduces an API to express vector computations**.**

2. Explain how sealed classes work and provide an example scenario where they would be beneficial.

Sealed classes in Java 17 allow you to define a class or interface with a restricted set of subclasses using the **sealed** keyword and **permits** clause. This feature is useful for maintaining a controlled and predictable set of subclasses, enhancing code maintainability and security.

Example scenario: In a banking system, you might want to restrict the types of bank accounts to specific ones like SavingsAccount and CheckingAccount. Sealed classes can enforce this restriction.

```
public sealed class BankAccount permits SavingsAccount, CheckingAccount {

   // common properties and methods

}

public final class SavingsAccount extends BankAccount {

   // specific properties and methods for savings account
}
public final class CheckingAccount extends BankAccount {
    // specific properties and methods for checking account
}
```

In this example, the BankAccount class is sealed, and only SavingsAccount and CheckingAccount are permitted to extend it, ensuring a controlled class hierarchy.

3. Write a code snippet using pattern matching for instanceof to simplify type casting.

Pattern matching for instanceof in Java 17 allows you to test and cast an object to a given type in a single expression, reducing boilerplate code and enhancing readability.

Example:

```
public class PatternMatchingExample {
    public static void main(String[] args) {
        Object obj = "Hello, World!";

        if (obj instanceof String s) {
            System.out.println(s.toUpperCase());
        } else {
            System.out.println("Not a string");
        }
    }
}
```

Here, instanceof checks if obj is a String. If true, s is automatically cast to String for use within the if block.

4. Describe what records are and how they can be used to simplify data carrier classes.

Records in Java 17 are a special kind of class designed to hold immutable data, automatically generating boilerplate code like constructors, getters, equals, hashCode, and toString methods. This simplifies data carrier classes.

Example:

```
public record Person(String name, int age) {}
```

The Person record automatically generates necessary methods, reducing boilerplate code. Records are immutable, making them suitable for representing unchangeable data.

5. Write a code snippet demonstrating the use of text blocks for multi-line strings.

Text blocks in Java 17 allow for multi-line string literals, making it easier to work with large text blocks. They are enclosed in triple double-quotes and preserve the text's formatting.

Example:

```
public class TextBlockExample {
public static void main(String[] args) {
String json = """
{
"name": "John Doe",
"age": 30,
"city": "New York"
}
""";
System.out.println(json);
}
}
```

6. Write a code snippet using switch expressions to handle multiple cases more concisely.

Switch expressions in Java 17 provide a concise way to handle multiple cases. Unlike traditional switch statements, they can return a value and be used in a more functional style.

Example:

```
public class SwitchExpressionExample {
public static void main(String[] args) {
String day = "MONDAY";
int dayNumber = switch (day) {
case "MONDAY", "FRIDAY", "SUNDAY" -> 6;
case "TUESDAY" -> 7;
case "THURSDAY", "SATURDAY" -> 8;
case "WEDNESDAY" -> 9;
default -> throw new IllegalArgumentException("Invalid day: " + day);
};
System.out.println("Day number is: " + dayNumber);
}
}
```

7. How has the Random Number Generation been enhanced in Java 17?

Answer: Java 17 introduced new interfaces and implementations for pseudorandom number generators (PRNGs) as part of JEP 356. This enhancement provides more flexibility and better support for stream-based programming.

Example:

```
import java.util.random.RandomGenerator;
import java.util.random.RandomGeneratorFactory;

public class RandomGeneratorExample {
    public static void main(String[] args) {
        RandomGenerator random = RandomGeneratorFactory.of("L64X128MixRandom").create();
        random.ints(5).forEach(System.out::println);
    }
}
```

Explanation:

RandomGenerator: A new interface for random number generators.

RandomGeneratorFactory: Provides methods to create instances of RandomGenerator.

8. What is the Vector API in Java 17?

Answer: The Vector API, introduced as a second incubator feature in Java 17, provides an API to express vector computations. It enables developers to write complex vector operations that can be optimized at runtime.

Example:

```
import jdk.incubator.vector.*;
public class VectorAPIExample {
    public static void main(String[] args) {
        VectorSpecies<Integer> SPECIES = IntVector.SPECIES_PREFERRED;
        int[] a = {1, 2, 3, 4};
        int[] b = {5, 6, 7, 8};
        int[] c = new int[4];

        IntVector va = IntVector.fromArray(SPECIES, a, 0);
        IntVector vb = IntVector.fromArray(SPECIES, b, 0);
        IntVector vc = va.add(vb);
        vc.intoArray(c, 0);

        for (int i : c) {
            System.out.println(i);
        }
    }
}
```

Explanation:

VectorSpecies: Describes the species of vector, which includes the element type and vector length.

IntVector: A concrete class that provides vector operations for integers.

9. What are the improvements in garbage collection in Java 17?

Answer: Java 17 includes several improvements in garbage collection, such as:

Improved G1 garbage collector performance and scalability.

Enhanced ZGC with lower latency and better support for large heaps.

Continued enhancements to the Shenandoah garbage collector.

10. What is the purpose of deprecating the Applet API in Java 17?

Answer: The Applet API has been deprecated for removal in Java 17 as part of JEP 398. The use of applets has declined significantly, and modern web technologies such as HTML5, CSS, and JavaScript have replaced them. Deprecating the Applet API simplifies the Java platform by removing obsolete features.

11. How Sealed classes are different from final classes?

Sealed classes and final classes serve different purposes in Java, although both are used to restrict inheritance.

Final Classes in Java cannot be inherited at all. It can be used when a class should never be extended.

Sealed Classes can be inherited, but only by a predefined set of classes. It is used when you want to allow inheritance, but only for specific classes.

12. Is it possible to use records with inheritance?

A record declaration does not have an extends clause, so it is not possible to explicitly declare a direct superclass type, even a Record. However, a record can implement interfaces, so you can use them polymorphically.